МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(МОСКОВСКИЙ ПОЛИТЕХ)**

09.03.01 Информатика и вычислительная техника

Образовательная программа (профиль)

«Интеграция и программирование в САПР»

Кафедра «СМАРТ технологии»

КУРСОВОЙ ПРОЕКТ

по дисциплине:

**Инженерное проектирование**

Разработка реляционной базы данных.

на тему:

Клапаны трубопроводной арматуры.

Преподаватель: / к.т.н. Толстиков А. В. /

*подпись ФИО, уч. звание и степень*

Студент: C:\Users\mirak\Downloads\2021-07-01_10-33-59.png / Калинин М. Д. 191-323 /

*подпись ФИО, группа*

Москва, 2021 г.

# ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

Задание №40

Разработать реляционную базу данных «Клапаны трубопроводной арматуры» на любом языке высокого уровня.

База данных должна содержать всю информацию, представленную в справочнике промышленного оборудования на страницах 41, 42, 45 ÷ 47, 70 ÷ 73, 76, 77, 80 и 81 (файл «Справочник Пром об 1»).

База данных должна содержать информацию о марке, назначении, условиях эксплуатации, технических характеристиках, запорных клапанов (вентилей). Информация о запорных клапанах может быть расширена, например, за счет чертежей и т.п. База данных должна быть расширена за счет клапанов аналогичных типов.

В БД должен быть обеспечен быстрый и удобный выбор клапанов по ряду параметров: марка, материал, номинальное давление, рабочая среда, температура окружающей среды, тип присоединения к трубопроводу, габаритные и присоединительные размеры, назначению, условиям эксплуатации. Параметры поиска могут быть расширены.

Создать комплекс форм, запросов, отчетов и меню, обеспечивающих удобную работу пользователя с базой данных. Обеспечить возможность дополнять, редактировать, удалять, сортировать, группировать данные, выполнять запросы, генерировать отчеты. Обеспечить защиту базы данных при изменении и добавлении данных.

# АННОТАЦИЯ

Калинин М. Д. Реляционная база данных «Клапаны трубопроводной арматуры».

Кафедра СМАРТ-технологий, 2021 г

Пояснительная записка – 33 стр.

Разработана реляционная база данных «Клапаны трубопроводной арматуры» с использованием СУБД MySQL. Разработано серверное приложение с использованием Node.js и фреймворка Express. Разработан веб-интерфейс с использованием фреймворка Angular.

База данных содержат всю информацию, представленную в справочнике промышленного оборудования на страницах 41, 42, 45 ÷ 47, 70 ÷ 73, 76, 77, 80 и 81 (файл «Справочник Пром об 1»).

База данных расширена за счет клапанов аналогичных типов. Реализован быстрый и удобный выбор клапанов по ряду параметров: марка, материал, номинальное давление, рабочая среда, температура окружающей среды, тип присоединения к трубопроводу, габаритные и присоединительные размеры, назначению, условиям эксплуатации.

Создан комплекс форм, запросов, отчетов и меню, обеспечивающих удобную работу пользователя с базой данных. Обеспечить возможность дополнять, редактировать, удалять, сортировать, группировать данные, выполнять запросы, генерировать отчеты. Обеспечить защиту базы данных при изменении и добавлении данных.

ОГЛАВЛЕНИЕ

[I. ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ 2](#_Toc75977932)

[II. АННОТАЦИЯ 3](#_Toc75977933)

[ОГЛАВЛЕНИЕ 4](#_Toc75977934)

[III. ВВЕДЕНИЕ 5](#_Toc75977935)

[3.1. Цель проекта 5](#_Toc75977939)

[3.2. Актуальность проекта 5](#_Toc75977940)

[3.3. Краткое содержание 5](#_Toc75977941)

[3.4. Задание 5](#_Toc75977942)

[IV. ОСНОВНАЯ ЧАСТЬ 6](#_Toc75977943)

[4.1. Постановка задачи 6](#_Toc75977945)

[4.2. Метод решения задачи 6](#_Toc75977946)

[4.3. Алгоритм решения задачи 6](#_Toc75977947)

[4.4. Программная реализация метода решения задачи 9](#_Toc75977948)

[4.4.1. Методы и средства разработки программного продукта 9](#_Toc75977949)

[4.4.2. Описание программного продукта 11](#_Toc75977950)

[4.4.3. Руководство системного программиста 17](#_Toc75977951)

[4.4.4. Руководство пользователя 19](#_Toc75977952)

[V. ЗАКЛЮЧЕНИЕ 21](#_Toc75977953)

[VI. СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 22](#_Toc75977954)

[VII. ПРИЛОЖЕНИЕ 23](#_Toc75977955)

# ВВЕДЕНИЕ

3. 1. **Цель проекта**

Целью работы является разработка реляционной базы данных «Клапаны трубопроводной арматуры» на языке программирования высокого уровня.

* 1. **Актуальность проекта**

Разработанная, в ходе курсового проекта база данных, актуальна для поиска необходимых клапанов трубопроводной арматуры. Также разработанная база данных может использоваться в образовательных целях в качестве справочника.

* 1. **Краткое содержание**

В основной части кратко описывается постановка задачи, метод решения, программная реализация.

* 1. **Задание**

Выполнить задание согласно варианту № 40. Разработать реляционную базу данных «Клапаны трубопроводной арматуры». Создать комплекс форм, запросов, отчетов и меню, обеспечивающих удобную работу пользователя с базой данных. Обеспечить возможность дополнять, редактировать, удалять, сортировать, группировать данные, выполнять запросы, генерировать отчеты. Обеспечить защиту базы данных при изменении и добавлении данных.

# ОСНОВНАЯ ЧАСТЬ

1. 1. **Постановка задачи**

Необходимо разработать реляционную базу данных «Клапаны трубопроводной арматуры» на любом языке программирования высокого уровня.

База данных должна содержать информацию о марке, назначении, условиях эксплуатации, технических характеристиках запорных клапанов (вентилей).

В базе данных должен быть осуществлен выбор клапана по ряду параметров: марка, материал, номинальное давление, рабочая среда, температура окружающей среды, тип присоединения к трубопроводу, габаритные и присоединительные размеры, назначению, условиям эксплуатации.

Разработать комплекс форм для добавления, удаления и редактирования информации о клапанах. Создать отчет с подробной информацией о клапане. Добавление и редактирование БД должно быть защищено.

* 1. **Метод решения задачи**

В качестве СУБД выбрана MySQL. Для взаимодействия с БД – веб-приложение.

Для веб-приложения необходимо разработать клиентскую и серверную часть.

Клиентская часть отвечает за взаимодействие с пользователем и отправку HTTP-запросов на сервер. Клиентская часть разрабатывается с помощью фреймворка Angular. Она будет запускаться в веб-браузере пользователя и будет представлять SPA (Single Page Application).

Серверная часть отвечает за получение HTTP-запросов от клиента и получение и редактирование данных в базе данных. Для разработки серверной части используется технология Node.js с фреймворком Express.

Аутентификация будет реализована с помощью Firebase. Добавление и редактирование будет доступно после ввода email и пароля администратора: **admin@mospolytech.ru**, **qwerty123**.

Для хранения изображений и статических файлов используется технология CDN (Content Delivery Network) и сервис Cloudinary.

Проект и БД выгружены в открытый доступ на облачном сервисе Heroku.

* 1. **Алгоритм решения задачи**

Перед созданием БД была спроектирована её структура, чтобы она соответствовала реляционной модели, в виде ER-диаграммы (рис. 1)

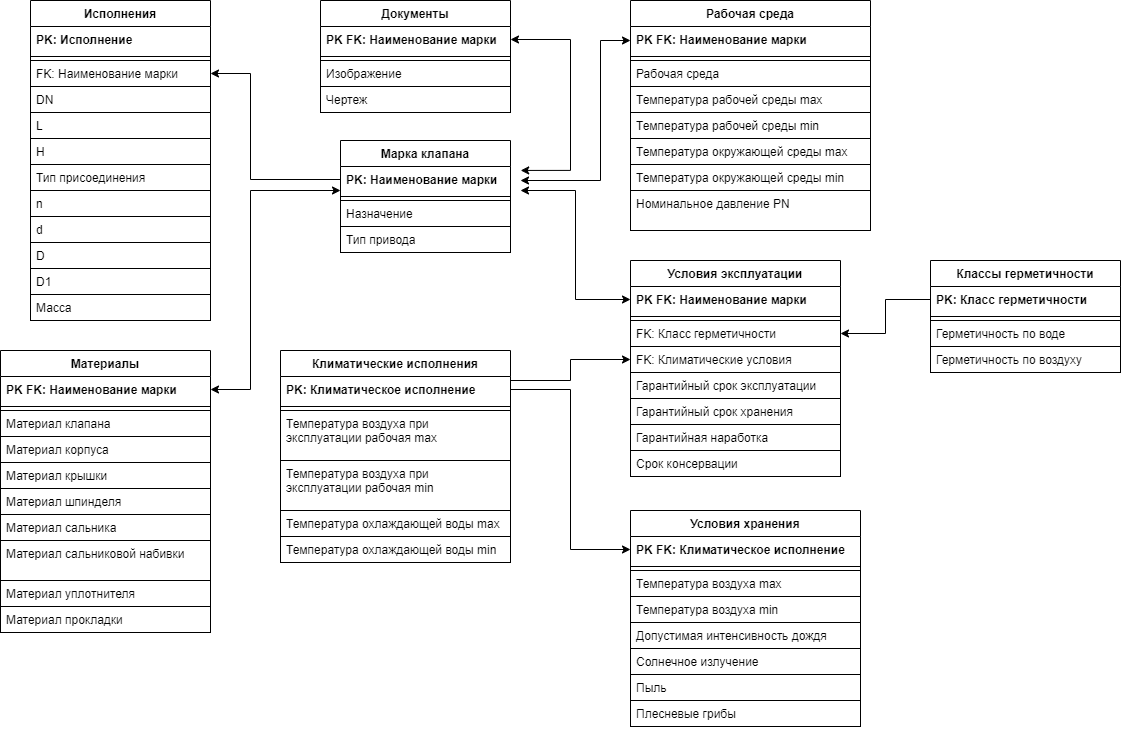


Рис. 1 - ER-диаграмма базы данных

После этого была созданы необходимые таблицы в СУБД MySQL и занесены данные в них. в них.

После этого была начата разработки сервера. Прежде всего были определены маршруты для HTTP-запросов (Табл. 1)

Таблица 1 – список HTTP-запросов к серверу и их описание

|  |  |  |
| --- | --- | --- |
| **Маршрут** | **Тип** | **Описание** |
| …/api/v1/valve | GET | Получение списка клапанов с краткой информацией |
| …/api/v1/valve/:mark | GET | Получение информации (кроме исполнений) об определенной марке клапана. |
| …/api/v1/valve/short | GET | Получение списка всех наименований марок |
| …/api/v1/valve/env | GET | Получение списка всех рабочих сред клапанов в базе данных |
| …/api/v1/valve/filter | GET | Получение списка клапанов, удовлетворяющим условиям, переданным в параметре запроса |
| …/api/v1/valve/:mark | DELETE | Удаляет марку клапана из БД |
| …/api/v1/valve/:mark | PUT | Редактирует информацию о марке клапана |
| …/api/v1/valve | POST | Добавляет марку клапана в БД |
| …/api/v1/execution/:mark | GET | Получение списка размеров исполнений марки клапана. |
| …/api/v1/execution/edit | PUT | Редактирует размеры в исполнений марки клапана. |
| …/api/v1/execution/delete | DELETE | Удаляет исполнение марки клапана. |
| …/api/v1/execution | POST | Добавляет исполнение марки клапана. |
| …/api/v1/ conditions/tightness | GET | Получение списка классов герметичности. |
| …/api/v1/ conditions/climate | GET | Получение списка климатических исполнений. |

Каждый маршрут выполняется в отдельном контроллере, который делает запрос к БД.

После этого были определены страницы (маршруты) интерфейса пользователя (табл. 2)

Таблица 2 – список страниц интерфейса пользователя

|  |  |
| --- | --- |
| **Маршрут** | **Описание страницы** |
| …/ | Главная страница с приветствием |
| …/login | Страница авторизации |
| …/valve/ | Страница со списком всех клапанов и краткой информации о каждом |
| …/valve/add | Страница с формой добавления марки клапана |
| …/valve/:mark/edit | Страница с формой редактирования марки клапана |
| …/valve/filter | Страница с формой подбора и поиска клапанов |
| …/valve/:mark | Страница с подробной информацией о марке клапана |
| …/execution/add | Страница с формой добавления исполнения для марки |
| …/execution/edit | Страница с формой редактирования исполнения для марки |

* 1. **Программная реализация метода решения задачи**
     1. **Методы и средства разработки программного продукта**

При проектировании базы данных была проанализирована предметная область после этого выявлены сущности и взаимосвязи между ними.

При разработке серверной части использовалась система маршрутизации (рис. 2) и контроллеры (рис. 3).

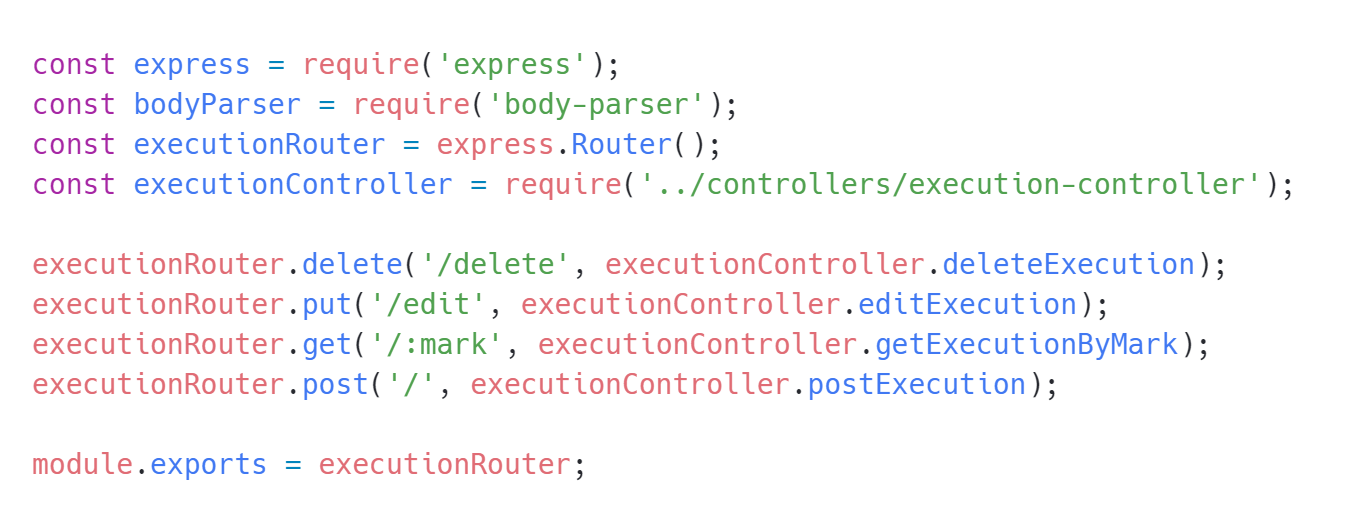


Рис. 2 – пример маршрута для исполнений.

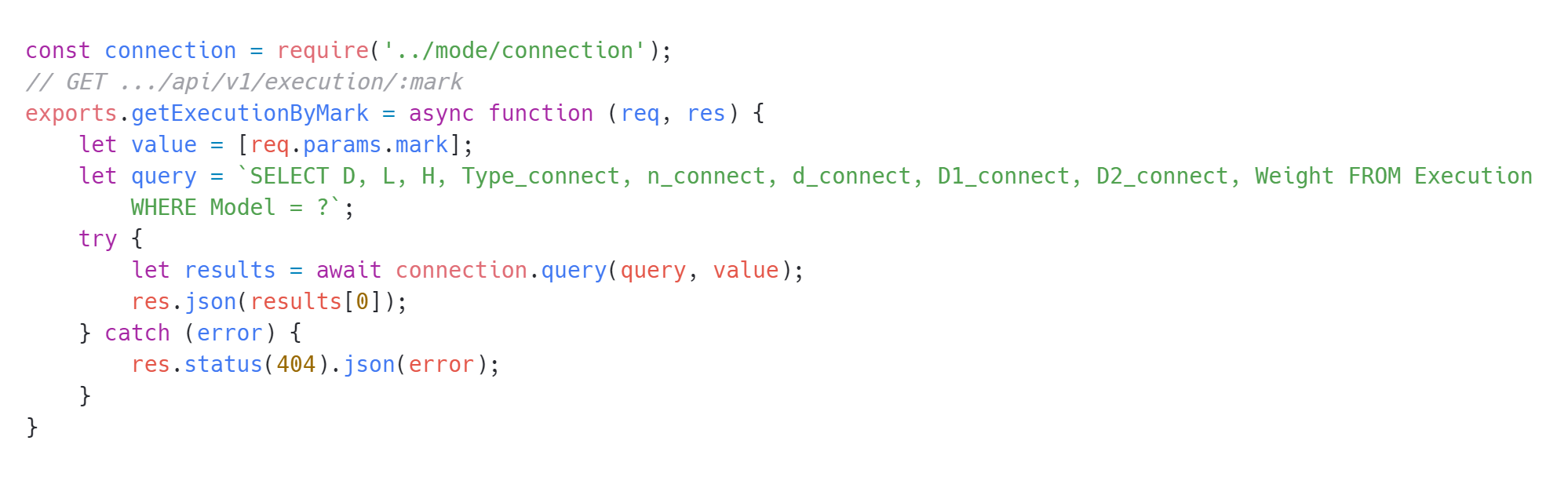


Рис. 3 – пример контроллера получения всех исполнений марки.

Также подключение к БД реализовывалось с помощью драйвера mysql2. В файл конфигурации (рис. 4) заносились данные для подключения. После эти данный использовались при создании пула подключения (рис. 5). После чего через созданный пул контроллеры могут делать запросы к БД.



Рис. 4 – файл конфигурации подключения к БД.



Рис. 5 – создание и экспорт пула подключения.

При разработке веб-интерфейса использовались компоненты Angular. Компоненты – это блоки, из которых состоит приложение. Каждый компонент имеет свой шаблон (рис. 6) и логику работы (рис. 7).

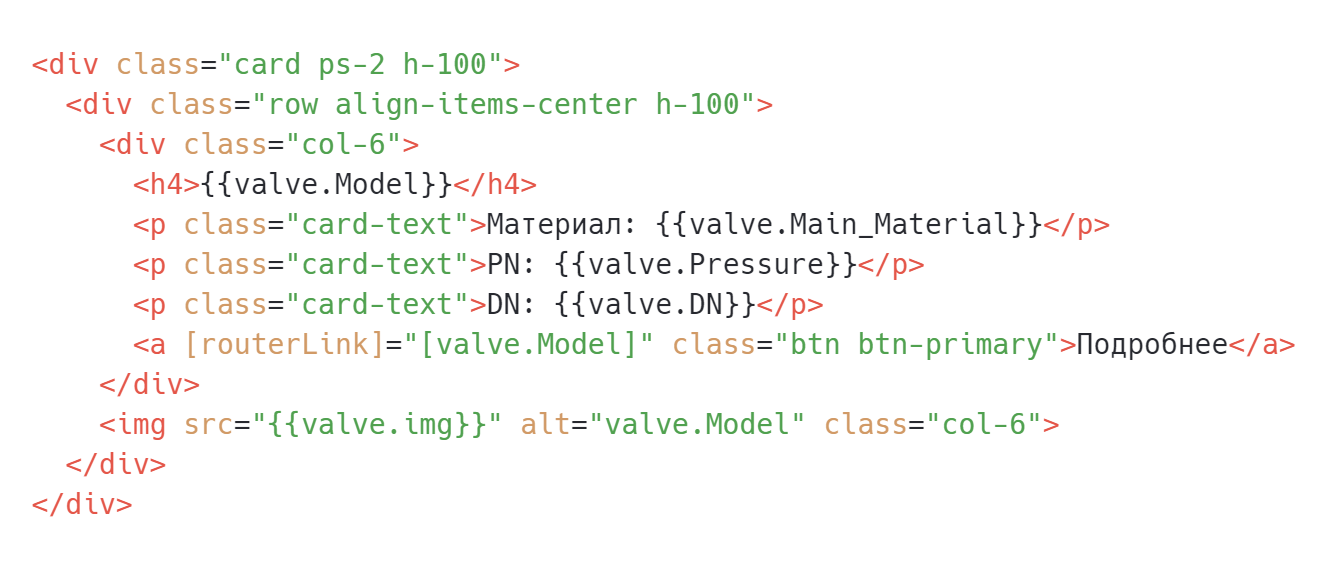


Рис. 6 – пример шаблона карточки клапана.



Рис. 7 – пример компонента карточки клапана.

Также использовались маршруты Angular для навигации по страницам. Для начала в корневом компоненте добавим тег <app-footer> для того чтобы на место этого тега подставлялся необходимый компонент в зависимости от маршрута (рис. 8).

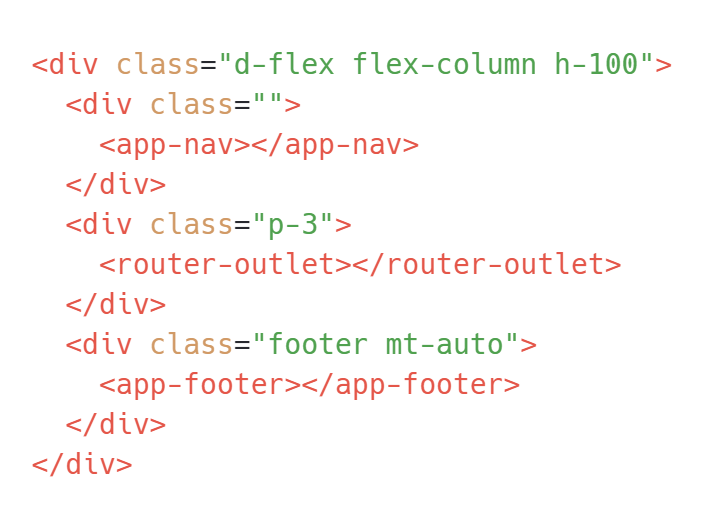


Рис. 8 – шаблон корневого компонента.

После этого пропишем маршруты для корневого компонента (рис. 9).



Рис. 9 – маршруты для корневого компонента.

* + 1. **Описание программного продукта**

**Введение**

Продукт представляет из себя веб-приложение с информацией о клапанах в базе данных. Помимо этого, пользователь также может добавлять и редактировать эту информацию.

**Структура программного обеспечения**

Программный продукт содержит несколько разделов:

1. Главный экран с приветствием (рис. 10) – приветствие и ссылки на список клапанов и добавление марки клапана.

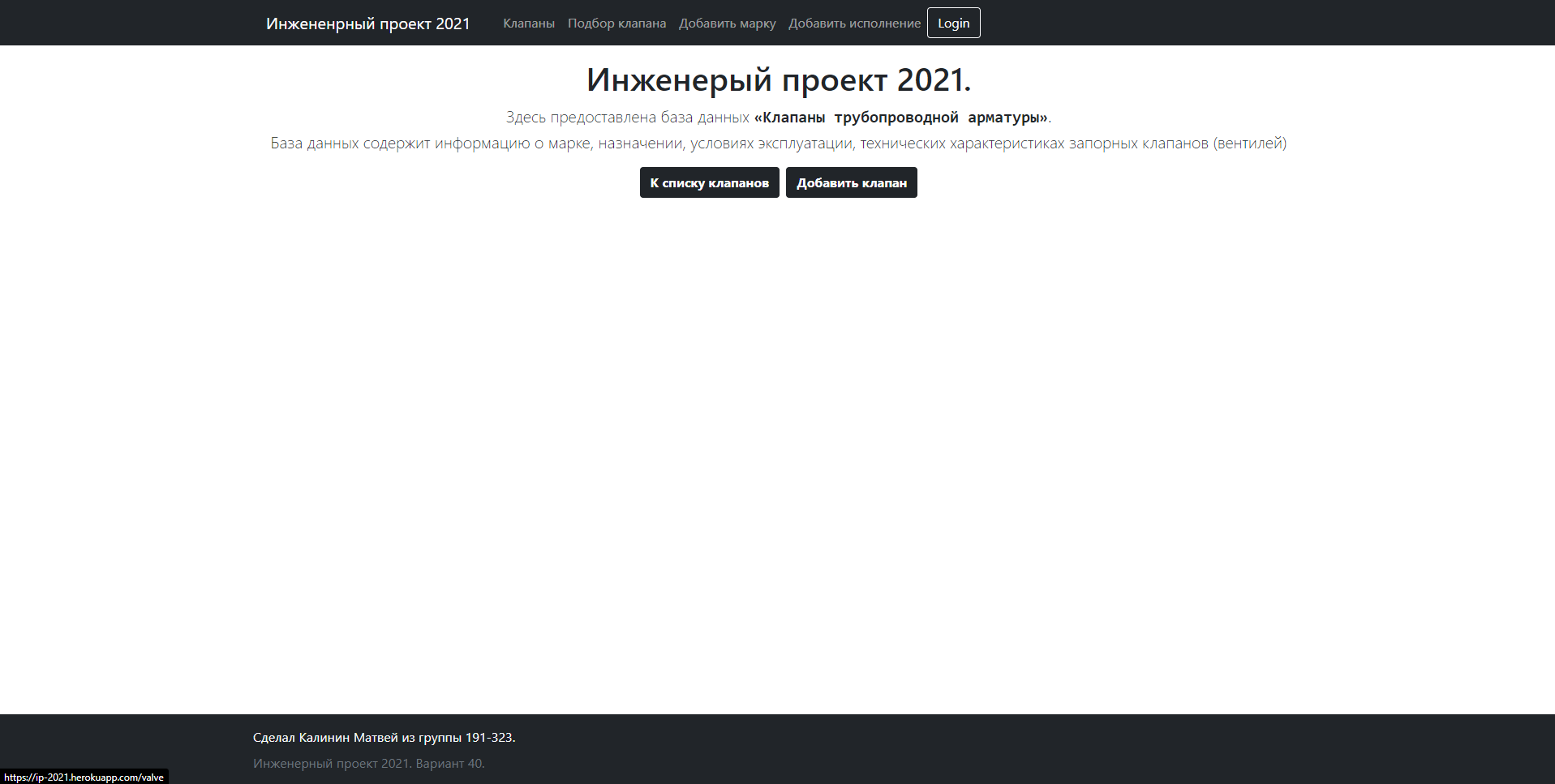


Рис. 10 – главный экран с приветствием.

1. Список клапанов (рис. 11) – содержит список всех клапанов в БД с краткой информацией о каждом (материал, номинальные диаметры, номинальное давление). Каждый клапан отображается в виде карточки и имеет кнопку для получения подробной информации о нем.

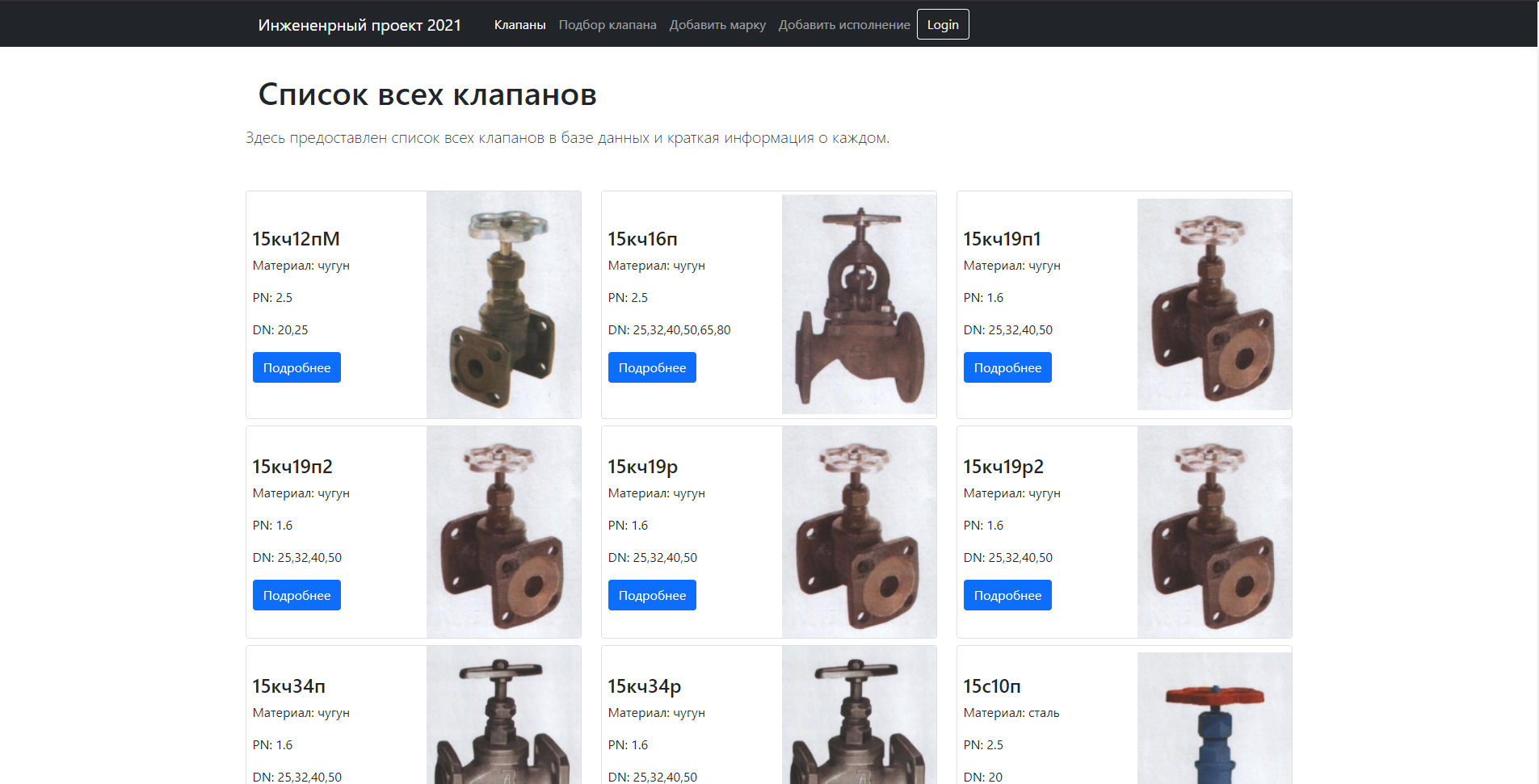


Рис. 11 – список клапанов.

1. Страница клапана (рис. 12) – отображает подробную информацию о клапане. Имеет кнопки для удаления и редактирования. Также для каждого исполнение клапана есть кнопки редактирования и удаления.

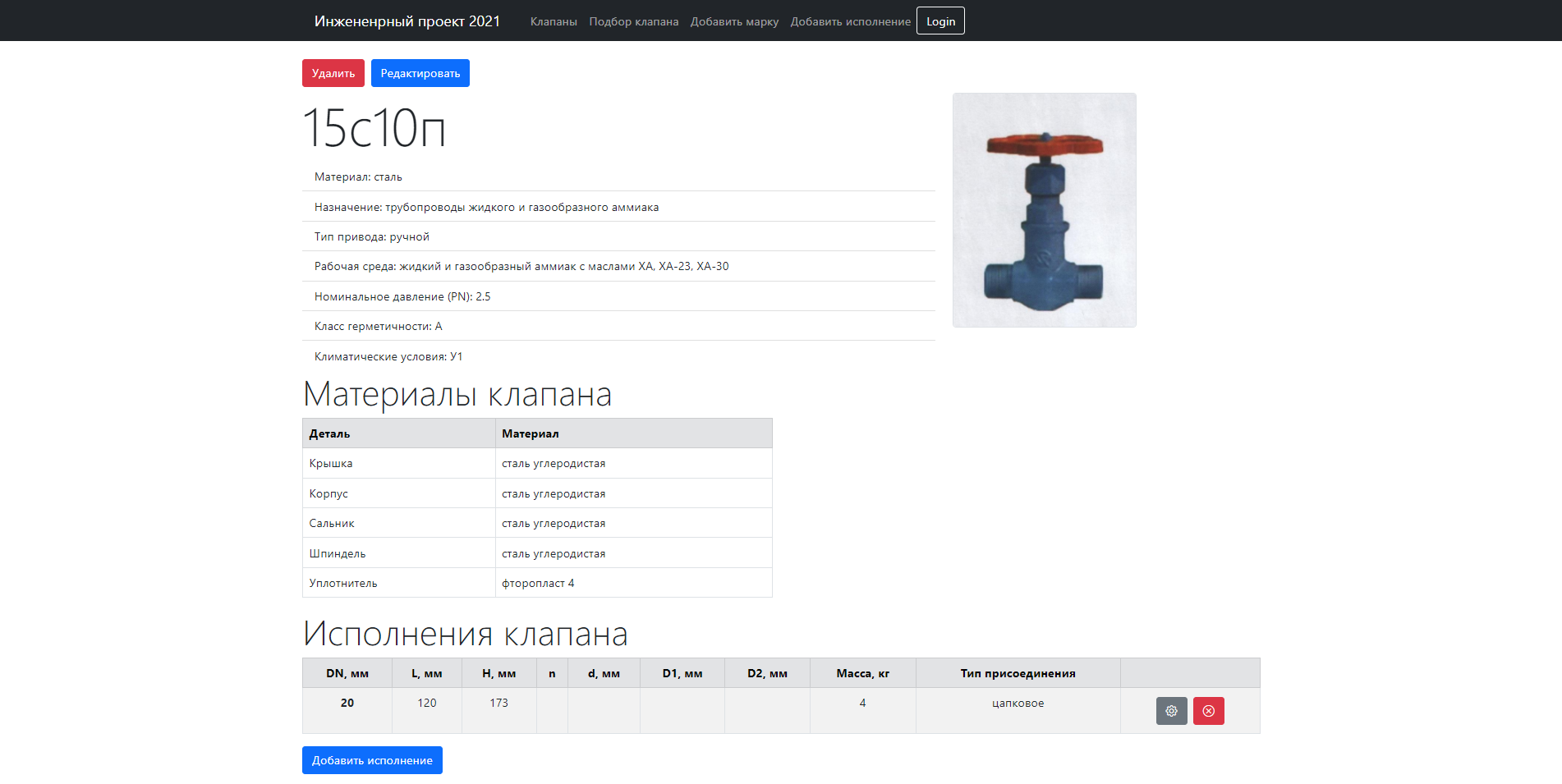


Рис. 12 – страница клапана.

1. Страница подбора и поиска клапана (рис. 13) – имеет поисковую строку и форму подбора клапана по номинальному диаметру, номинальному давлению, температуре окружающей среды, присоединению, материалу, рабочей среде, классу герметичности и климатическому исполнению.

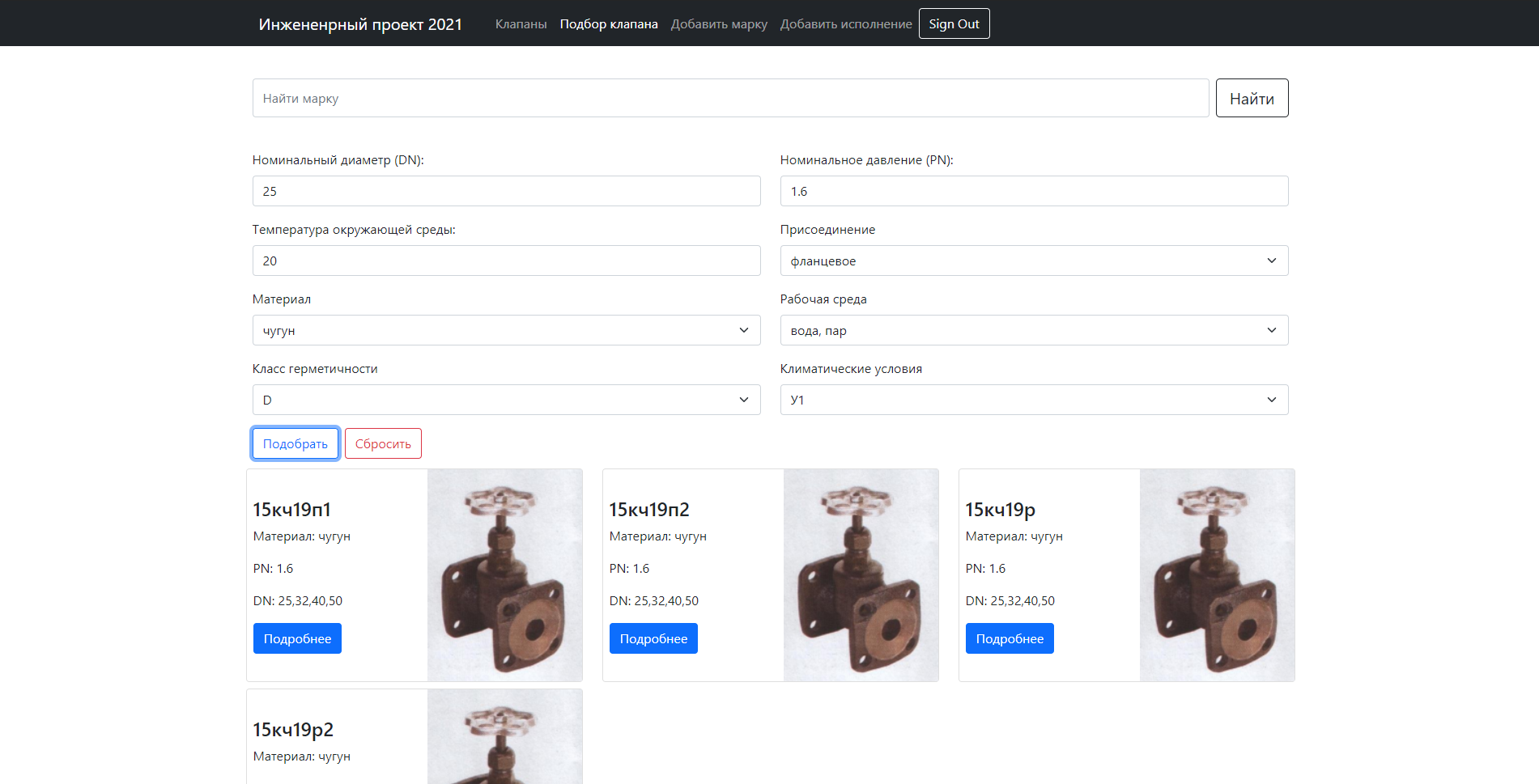


Рис. 13 – страница поиска и подбора клапана.

1. Страница авторизации (рис. 14) – имеет форму авторизации. Позволяет авторизоваться администратору и вносить изменения в БД.

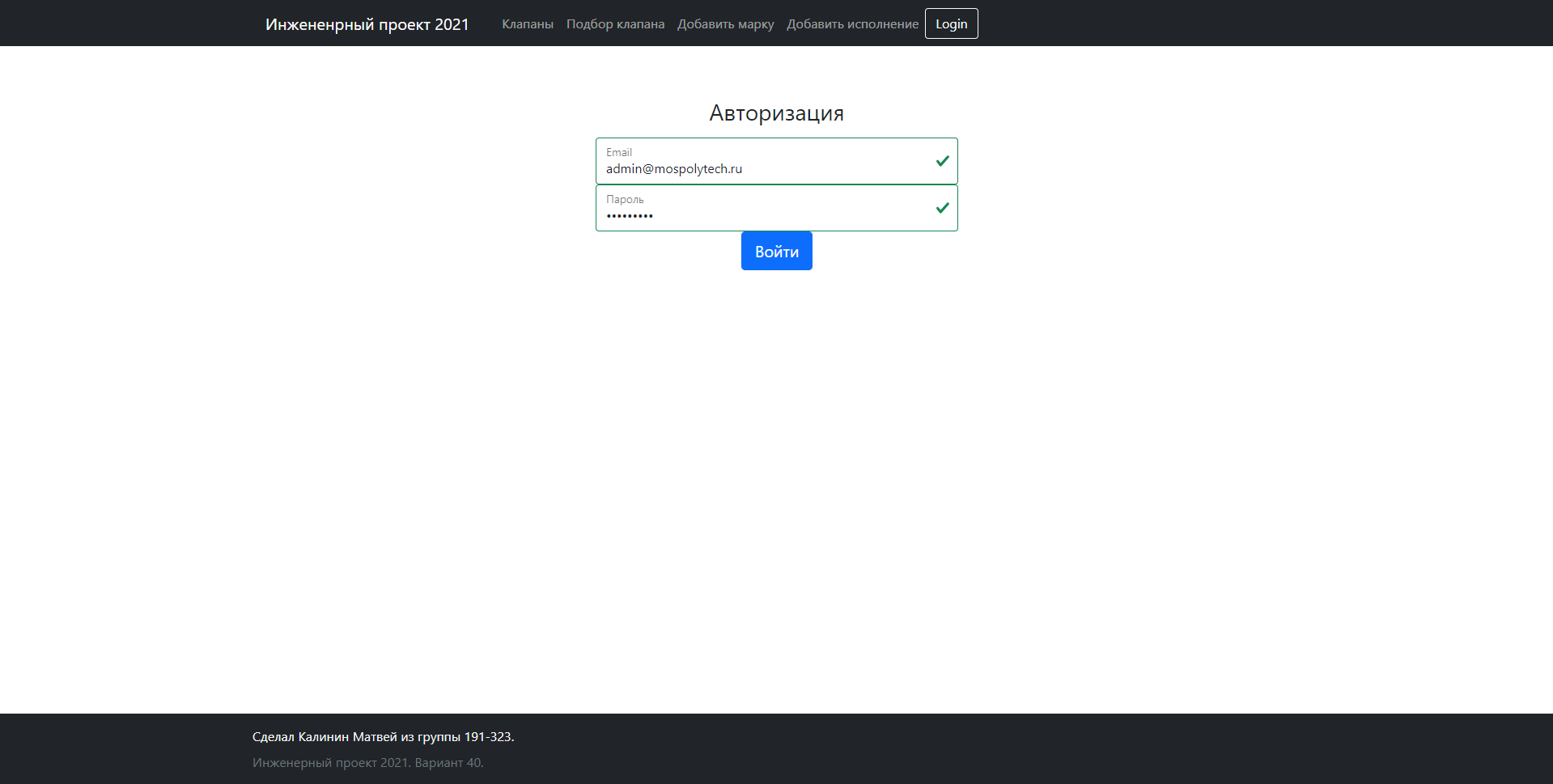


Рис. 14 – страница авторизации.

1. Страница добавления марки клапана (рис. 15) – имеет форму добавления марки клапана. Доступна только администратору.

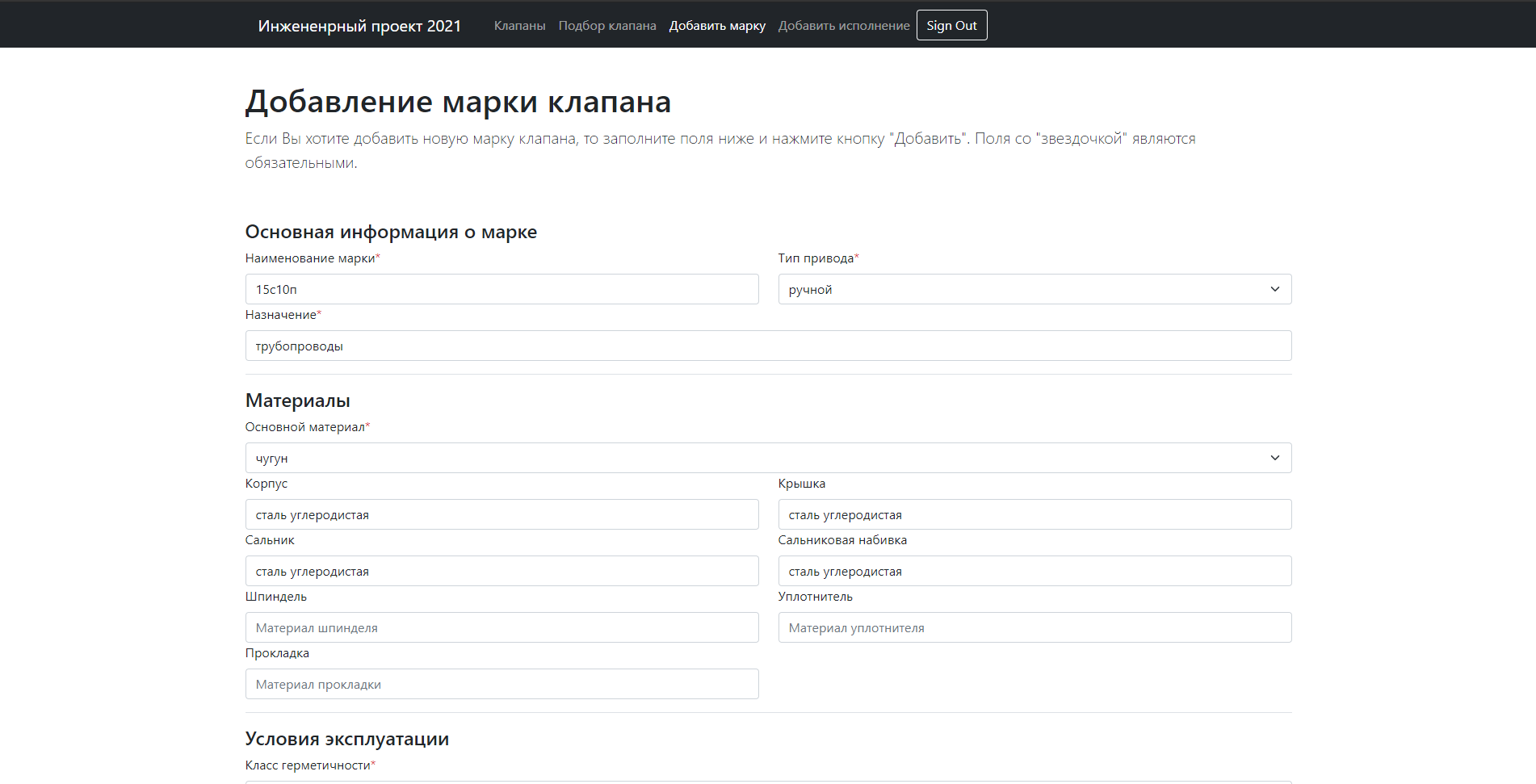


Рис. 15 – страница добавления марки клапана.

1. Страница добавления исполнения марки клапана (рис. 16) – имеет форму для добавления исполнения для марки клапана. Доступна только администратору.

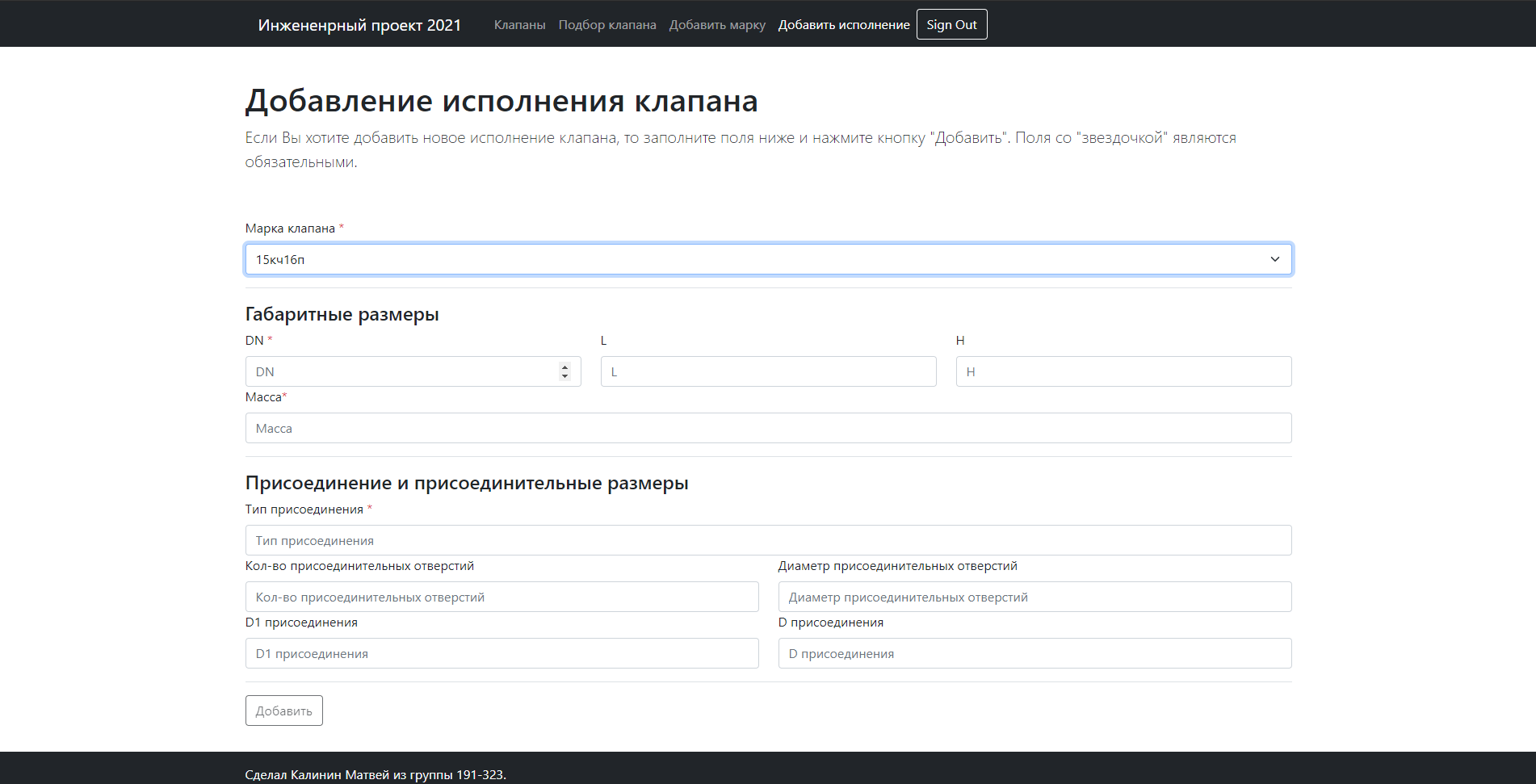


Рис. 16 – страница добавления исполнения марки клапана.

1. Страница редактирования марки клапана (рис. 17) – имеет форму редактирования марки клапана. Доступна только администратору.

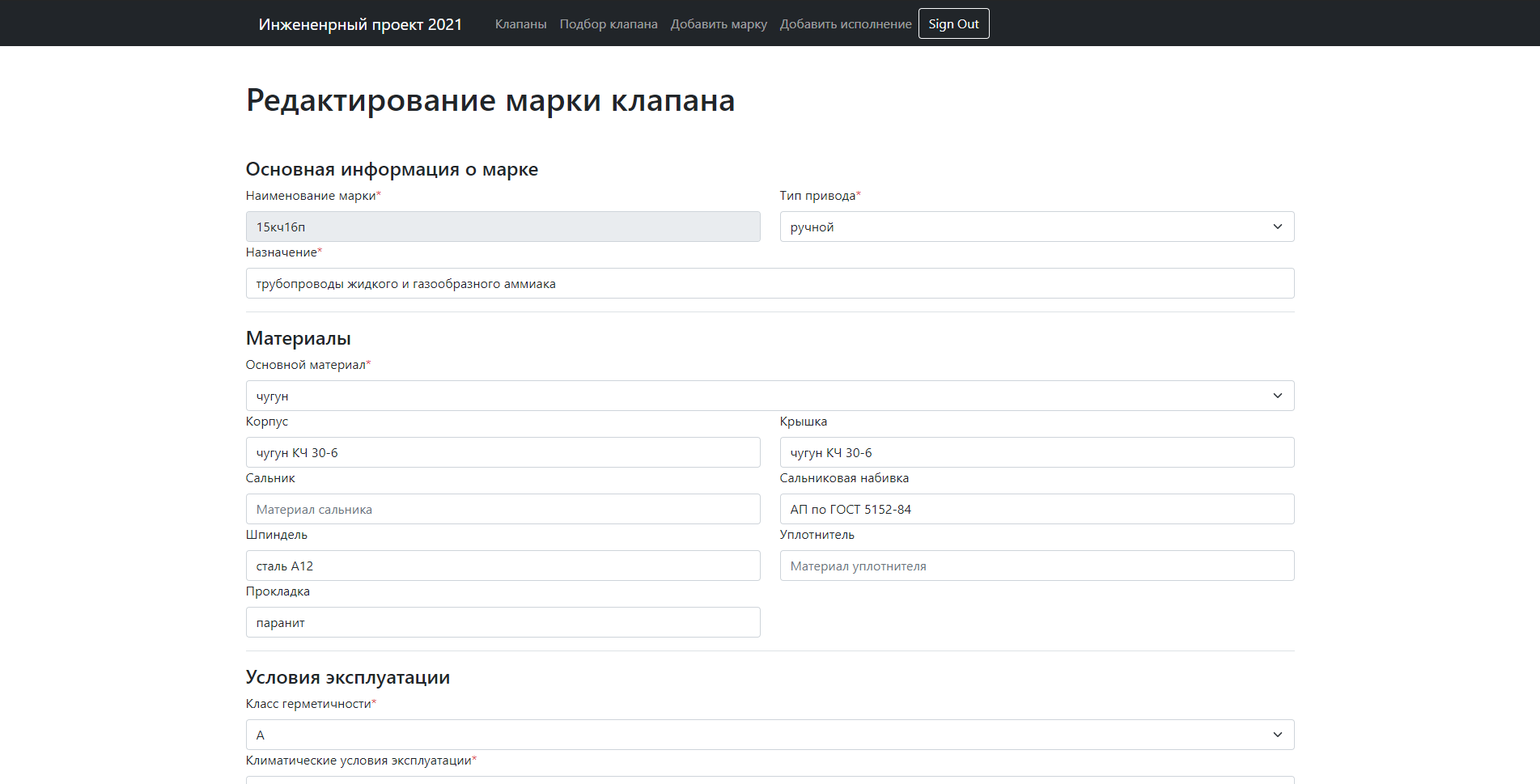


Рис. 17 – страница редактирования марки клапана.

1. Страница редактирования исполнения марки клапана (рис. 18) – имеет форму для редактирования исполнения для марки клапана. Доступна только администратору.

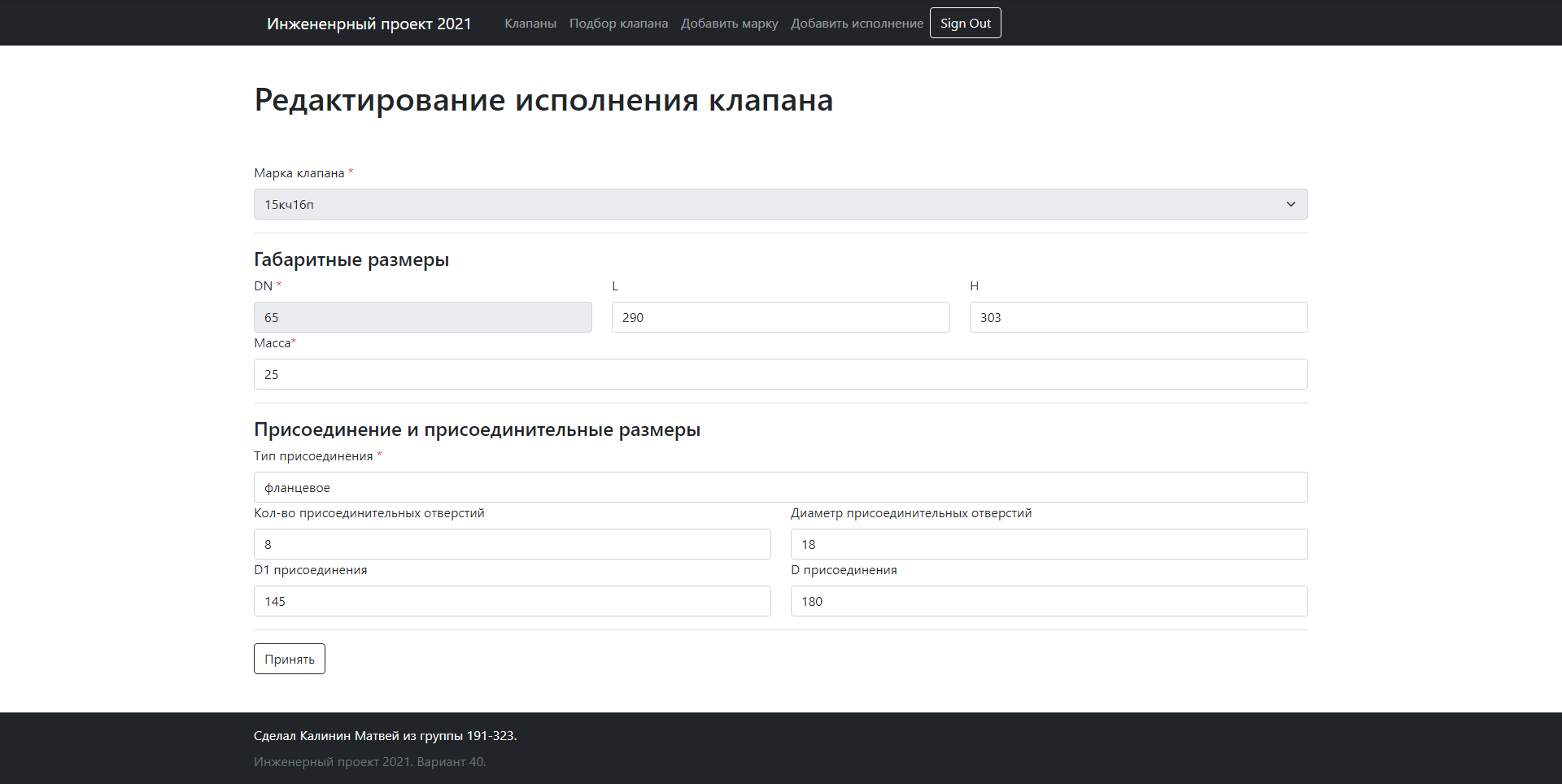


Рис. 18 – страница редактирования исполнения марки клапана.

Помимо этого, на каждой странице есть навигационная панель для более удобного перехода между страницами. Также есть «подвал» страницы, где расположена информация о продукте.

**Описание работы программы**

Поскольку продукт представляет собой веб-приложение, то оно работает по принципу сервера: ожидает входящие запросы, обрабатывает их (при необходимости делает запросы в базу данных) и отправляет ответ.

Сперва пользователь в веб-браузере переходит по ссылке <https://ip-2021.herokuapp.com/>. В ответ сервер отправляет ему статические данные (html файлы, скрипты, стили и т.п.) для отображения страницы. После этого пользователь переходит в интересующие его разделы. Если нужно получить данные из БД или произвести там какие-либо действия, то из веб-браузера отправляется HTTP-запрос на сервер. Сервер получает его и с помощью системы маршрутизации определяет на какой контроллер передать запрос. Контроллер делает запросы к БД, обрабатывает результаты и отправляет ответ обратно в веб-браузер.

**Сведения об операционной системе**

Продукт полностью независим от операционной системы Клиентская часть работает в веб-браузере, поэтому никак не зависит от операционной системы. Серверная часть работает на кроссплатформенной технологии Node.js.

* + 1. **Руководство системного программиста**

**Назначения и условия**

Для использования продукта необходим компьютер с установленным веб браузером. Технические характеристики компьютера должны соответствовать техническим требованиям вашего веб-браузера.

Назначение: поиск клапанов трубопроводной арматуры и использование в качестве справочника.

**Структура программы**

База данных располагаемся на удаленном хостинге CleanDB и все запросы отправляются на него. Сервер расположен на облачном сервисе Heroku.

Аутентификация происходит с помощью облачного сервиса Firebase.

Получение изображений и файлов клапанов происходит с использованием технологий CDN, а именно сервиса Cloudinary. В БД хранится ссылка на файл или изображение, а веб-браузер сам загружает его.

**Характеристики программы**

* Среднее время запроса к серверу БД – 160 мс.
* Среднее время загрузки изображения – 25 мс.
* Среднее время авторизации – 370 мс.

**Настройка программы**

Настройка программы не требуется.

**Входные и выходные данные**

Поскольку общение клиента и сервера происходит с помощью HTTP-запросов, то входные данные будут передаваться в теле запроса, выходные данные – в теле ответа.

Таблица 3 – данные запроса и ответа к серверу.

|  |  |  |
| --- | --- | --- |
| Запрос | Входные данные | Выходные данные |
| Добавление марки клапана | Тело запроса: | Успешный ответ:    Ответ ошибки:  Строка сообщения ошибки |
| Добавление исполнения марки клапана | Тело запроса: | Успешный ответ:    Ответ ошибки:  Строка сообщения ошибки |
| Редактирование исполнения клапана | Тело запроса: | Успешный ответ:    Ответ ошибки:  Строка сообщения ошибки |
| Редактирование марки клапана | Тело запроса: | Успешный ответ:    Ответ ошибки:  Строка сообщения ошибки |

**Сообщения системному программисту**

Сообщения системному программисту приходят в виде кодов статуса HTTP-запросов:

* 200 – успешный запрос.
* 201 – в результате успешного выполнения запроса был создан новый ресурс.
* 400 – сервер обнаружил в запросе клиента ошибку.
* 404 – ресурс не найден
* 401 – для доступа к запрашиваемому ресурсу требуется аутентификация.
* 500 – любая внутренняя ошибка сервера.
  + 1. **Руководство пользователя**

**Назначение программы**

Поиск клапанов трубопроводной арматуры и использование в качестве справочника.

**Условия применения программы**

Для применения программы на компьютере должен быть установлен веб-браузер и стабильное подключение к интернету.

**Пуск программы**

Для запуска программы необходимо в строке адреса веб-браузера ввести: <https://ip-2021.herokuapp.com/>.

**Команды пользователя**

Для навигации пользователя в верней части экрана есть навигационная панель с кнопками для перехода в соответствующие разделы. Помимо этого, в необходимых местах присутствуют навигационные кнопки.

1. ЗАКЛЮЧЕНИЕ

В результате была разработана реляционная база данных «Клапаны трубопроводной арматуры».

Также разработано веб-приложение для взаимодействия с базой данных. Обеспечен быстрый и удобный выбор клапанов по ряду параметров: марка, материал, номинальное давление, рабочая среда, температура окружающей среды, тип присоединения к трубопроводу, условиям эксплуатации. Создан комплекс форм, запросов, отчетов и меню, обеспечивающих удобную работу пользователя с базой данных. Обеспечена возможность дополнять, редактировать, удалять. Обеспечена защита базы данных при изменении и добавлении данных с помощью авторизации.

Проект выгружен в удаленный хостинг, чтобы был доступен для пользования в интернете.

1. СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ
2. *Журнал* справочник промышленного оборудования: трубопроводная арматура / изд. ЗАО «Вентиляция, водоснабжение, теплоснабжение»; гл. ред. А. Ю. Лезин – Москва, Шоссе Энтузиастов, 56, стр. 22 – ОАО «АСТ – Московский полиграфический дом», 2005 – 187 стр.
3. *ГОСТ 15150-69.* Машины, приборы и другие технические изделия: Исполнения для различных климатических районов: нац. стандарт Рос. Федерации: изд. офиц.: утв. и введ. в действие Постановлением Государственного комитета СССР по стандартам от 29.12.69 N 1394: введ. впервые: дата введ. 1971-01-01/ разраб. М.Л.Оржаховский (руководитель разработки); З.С.Боголюбова; Г.В.Козлова, канд. техн. наук; И.П.Меллер; М.С.Пинзур; Е.А.Судьин; B.М.Строганова; Г.П.Стрелкова; Г.Н.Трубецкая – Москва: Стандартинформ, 2010. – 76 стр.
4. *ГОСТ 9544-93.* Арматура трубопроводная запорная: нормы герметичности затворов: нац. стандарт Рос. Федерации: изд. офиц.: утв. и введ. в действие Межгосударственным Советом по стандартизации, метрологии и сертификации 21 октября 1993 г.: введ. впервые: дата введ. 1995-01-01/ разраб. Госстандартом России – Москва: Издательство стандартов, 1995. – 76 стр.
5. *Документация* фреймворка Express: официальный сайт. – США, 2017. – URL: <https://expressjs.com> (дата обращения 09.05.2021).
6. *Документация* фреймворка Angular: официальный сайт. – США, 2010. – URL: <https://angular.io/docs> (дата обращения 04.05.2021).
7. ПРИЛОЖЕНИЕ

Листинг 1 – SQL запросы создания таблиц

Листинг 2 – конфигурация для подключения в БД

module.exports = {

    host: 'eu-cdbr-west-01.cleardb.com',

    user: 'bd5a6e6d7bb6ed',

    password: '0d8a303e',

    database: 'heroku\_9e326092b46156b',

};

Листинг 3 – подключение к БД

const mysql2 = require('mysql2');

const dbConfig = require('../config/db.config');

const connection = mysql2.createPool(dbConfig).promise();

module.exports = connection;

Листинг 4 – запуск сервера Express

const express = require('express');

const cors = require('cors');

const morgan = require('morgan');

const app = express();

app.use(express.json());

app.use(cors());

app.use(morgan('dev'));

app.use(express.static('public'));

const valveRouter = require('./routes/valve-router'); //для адресов /valve

const conditionsRouter = require('./routes/conditions-router');

const executionRouter = require('./routes/execution-router');

app.use('/api/v1/valve', valveRouter);

app.use('/api/v1/conditions', conditionsRouter);

app.use('/api/v1/execution', executionRouter);

// обработка ошибки 404

app.use(function (req, res, next) {

    res.status(404).send("Not Found")

});

const port = process.env.PORT || 3000;

app.listen(port, () => {

    console.log(`http://localhost:${port}`);

});

Листинг 5 – обработка маршрута valve

const express = require('express');

const bodyParser = require('body-parser');

const valveRouter = express.Router();

const valveController = require('../controllers/valve-controller');

const urlencodedParser = bodyParser.urlencoded({

    extended: false

});

valveRouter.get('/', valveController.getValves);

valveRouter.get('/short', valveController.getValvesShort);

valveRouter.get('/env', valveController.getValveWorkEnv);

valveRouter.get('/filter', valveController.getValveFilter);

valveRouter.get('/:mark', valveController.getValveByMark);

valveRouter.delete('/:mark', valveController.deleteValve);

valveRouter.put('/:mark', valveController.updateValve);

valveRouter.post('/', urlencodedParser, valveController.addValve);

module.exports = valveRouter;

Листинг 6 – обработка маршрута execution

const express = require('express');

const bodyParser = require('body-parser');

const executionRouter = express.Router();

const executionController = require('../controllers/execution-controller');

executionRouter.delete('/delete', executionController.deleteExecution);

executionRouter.put('/edit', executionController.editExecution);

executionRouter.get('/:mark?', executionController.getExecutionByMark);

executionRouter.post('/', executionController.postExecution);

module.exports = executionRouter;

Листинг 7 – обработка маршрута conditions

const express = require('express');

const conditionsRouter = express.Router();

const conditionsController = require('../controllers/conditions-controller');

conditionsRouter.get('/tightness', conditionsController.getTightnessClass);

conditionsRouter.get('/climate', conditionsController.getClimateClass);

module.exports = conditionsRouter;

Листинг 8 – контроллер valve

const connection = require("../mode/connection");

// GET ../valve

exports.getValves = async function (req, res) {

    let query = `SELECT \* FROM Valve\_Model

    JOIN materials ON materials.Model = valve\_model.Model

    JOIN Work\_Enviroment ON Work\_Enviroment.Model = valve\_model.Model

    JOIN documents ON documents.Model = valve\_model.Model`;

    let results = (await connection.query(query))[0];

    let execution = (await connection.query('SELECT Model, D FROM execution'))[0];

    execution.forEach(item => {

        let i = results.findIndex(valve => valve.Model == item.Model);

        if (!('DN' in results[i])) {

            results[i]['DN'] = [];

        }

        results[i].DN.push(item.D);

    });

    res.status(200).json(results);

}

// GET ../valve/:mark

exports.getValveByMark = async function (req, res) {

    let query = `SELECT \* FROM Valve\_Model

    JOIN materials ON materials.Model = valve\_model.Model

    JOIN Work\_Enviroment ON Work\_Enviroment.Model = valve\_model.Model

    JOIN documents ON documents.Model = valve\_model.Model

    JOIN operating\_conditions ON operating\_conditions.Model = valve\_model.Model

    WHERE valve\_model.Model = ?`;

    let value = [req.params.mark];

    try {

        let results = await connection.query(query, value);

        res.json(results[0][0]);

    } catch (error) {

        res.json(error);

        return;

    }

}

// GET ..valve/short

exports.getValvesShort = async function (req, res) {

    let query = `SELECT Model FROM Valve\_Model`;

    try {

        let results = await connection.query(query);

        res.json(results[0].map(item => item.Model));

    } catch (error) {

        res.json(error);

        return;

    }

}

//GET ..valve/env

exports.getValveWorkEnv = async function (req, res) {

    let query = `SELECT Work\_Enviroment FROM work\_enviroment group by Work\_Enviroment`;

    try {

        let results = await connection.query(query);

        res.json(results[0].map(item => item.Work\_Enviroment));

    } catch (error) {

        res.json(error);

        return;

    }

}

// GET ..valve/filter

exports.getValveFilter = async function (req, res) {

    const queryParam = req.query;

    console.log(queryParam);

    const query = `SELECT Valve\_Model.Model, materials.Main\_Material, Work\_Enviroment.Pressure, documents.img FROM Valve\_Model

        JOIN materials ON

            materials.Main\_Material = '${queryParam.material}'

            AND materials.Model = Valve\_Model.Model

        JOIN work\_enviroment ON

            work\_enviroment.t\_env\_max >= ${queryParam.tEnw}

            AND work\_enviroment.t\_env\_min <= ${queryParam.tEnw}

            AND work\_enviroment.Work\_Enviroment = '${queryParam.workEnv}'

            AND work\_enviroment.Pressure = ${queryParam.PN}

            AND work\_enviroment.Model = Valve\_Model.Model

        JOIN documents ON

            documents.Model = valve\_model.Model

        JOIN execution ON

            execution.D = ${queryParam.DN}

            AND execution.Type\_connect = '${queryParam.typeConnect}'

            AND execution.Model = valve\_model.Model

        JOIN operating\_conditions ON

            operating\_conditions.climate\_conditions = '${queryParam.climateCondition}'

            AND operating\_conditions.tightness\_class = '${queryParam.classTightness}'

            AND operating\_conditions.Model = valve\_model.Model;`

    try {

        let results = (await connection.query(query))[0];

        let execution = (await connection.query('SELECT Model, D FROM execution'))[0];

        // console.log(execution)

        execution.forEach(item => {

            let i = results.findIndex(valve => valve.Model == item.Model);

            if (i != -1) {

                if (!('DN' in results[i])) {

                    results[i]['DN'] = [];

                }

                results[i].DN.push(item.D);

            }

        });

        res.status(200).json(results);

    } catch (error) {

        res.status(404).json(error);

        return;

    }

}

// DELETE ../valve/:mark

exports.deleteValve = async function (req, res) {

    const mark = req.params.mark;

    try {

        await connection.query(`DELETE FROM materials WHERE materials.Model = ?`, [mark]);

        await connection.query(`DELETE FROM work\_enviroment WHERE work\_enviroment.Model = ?`, [mark]);

        await connection.query(`DELETE FROM documents WHERE documents.Model = ?`, [mark]);

        await connection.query(`DELETE FROM operating\_conditions WHERE operating\_conditions.Model = ?`, [mark]);

        await connection.query(`DELETE FROM execution WHERE execution.Model = ?`, [mark]);

        await connection.query(`DELETE FROM Valve\_Model WHERE Valve\_Model.Model = ?`, [mark]);

        res.status(200).json({

            succes: true

        });

    } catch (error) {

        res.status(404).json(error);

        return;

    }

}

// PUT ../valve/:mark

exports.updateValve = async function (req, res) {

    const mark = req.params.mark;

    const values = req.body;

    try {

        await connection.query(`UPDATE valve\_model SET Purpose=?, Type\_drive=? WHERE Model=?`, [values.purpose, values.typeDrive, mark]);

        await connection.query(`UPDATE materials SET

        Main\_Material = ?,

        Cap\_Material = ?,

        Body\_Material = ?,

        OilSeal\_Material = ?,

        OilSealPack\_Material = ?,

        Spindle\_Material = ?,

        Sealer\_Material = ?,

        Gasket\_Material = ?

        WHERE Model=?`, [

            values.mainMaterial,

            values.capMaterial,

            values.bodyMaterial,

            values.oilSealMaterial,

            values.oilSealPackMaterial,

            values.spindleMaterial,

            values.sealerMaterial,

            values.gasketMaterial,

            mark

        ]);

        await connection.query(`UPDATE work\_enviroment SET

        Work\_Enviroment = ?,

        t\_work\_env\_max = ?,

        t\_work\_env\_min = ?,

        t\_env\_max = ?,

        t\_env\_min = ?,

        Pressure = ?

        WHERE Model=?`, [

            values.workEnv,

            values.tWorkMax,

            values.tWorkMin,

            values.tEnvMax,

            values.tEnvMin,

            values.pressure,

            mark

        ]);

        await connection.query(`UPDATE operating\_conditions SET

        tightness\_class = ?,

        climate\_conditions = ?,

        warranty\_operation = ?,

        warranty\_storage = ?,

        warranty\_time = ?,

        conservation = ?

        WHERE Model=?`, [

            values.tightnessClass,

            values.operatingConditions,

            values.warrantyOperation,

            values.warrantyStorage,

            values.warrantyTime,

            values.conservation,

            mark

        ]);

        await connection.query(`UPDATE documents SET img =? WHERE Model=?`, [values.img, mark]);

        res.status(200).json({

            succes: true

        });

    } catch (error) {

        res.status(404).json(error);

        return;

    }

}

// POST ../valve/

exports.addValve = async function (req, res) {

    let requestBody = req.body;

    //добавление в таблицу Valve\_model

    let query = `INSERT into Valve\_model (Model, Purpose, Type\_drive)

        values (?, ?, ?);`;

    let values = [

        requestBody.mark,

        requestBody.purpose,

        requestBody.typeDrive

    ];

    try {

        await connection.query(query, values);

        res.json({

            succes: true

        });

    } catch (error) {

        console.log(error);

        res.status(400).json(error);

        return;

    }

    //добавление в таблицу Work\_Enviroment

    query = `INSERT into Work\_Enviroment

    (Model, Work\_Enviroment, t\_work\_env\_min, t\_work\_env\_max, t\_env\_min, t\_env\_max, Pressure)

        values (?, ?, ?, ?, ?, ?, ?);`;

    values = [

        requestBody.mark,

        requestBody.workEnv,

        requestBody.tWorkMin,

        requestBody.tWorkMax,

        requestBody.tEnvMin,

        requestBody.tEnvMax,

        requestBody.pressure

    ];

    try {

        await connection.query(query, values);

    } catch (error) {

        console.log(error);

        res.status(400).json(error);

        return;

    }

    //добавление в таблицу Materials

    query = `INSERT into Materials

    (Model, Main\_Material, Cap\_Material, Body\_Material, OilSeal\_Material, OilSealPack\_Material, Spindle\_Material, Sealer\_Material, Gasket\_Material)

        values (?,?,?,?,?,?,?,?,?);`;

    values = [

        requestBody.mark,

        requestBody.mainMaterial,

        requestBody.capMaterial,

        requestBody.bodyMaterial,

        requestBody.oilSealMaterial,

        requestBody.oilSealPackMaterial,

        requestBody.spindleMaterial,

        requestBody.sealerMaterial,

        requestBody.gasketMaterial

    ];

    try {

        await connection.query(query, values);

    } catch (error) {

        console.log(error);

        res.status(400).json(error);

        return;

    }

    //добавление в таблицу operating\_conditions

    query = `INSERT into operating\_conditions

    (Model, tightness\_class, climate\_conditions, warranty\_operation, warranty\_storage, warranty\_time, conservation)

        values (?,?,?,?,?,?,?);`;

    values = [

        requestBody.mark,

        requestBody.tightnessClass,

        requestBody.operatingConditions,

        requestBody.warrantyOperation,

        requestBody.warrantyStorage,

        requestBody.warrantyTime,

        requestBody.conservation,

    ];

    try {

        await connection.query(query, values);

    } catch (error) {

        console.log(error);

        res.status(400).json(error);

        return;

    }

    //добавление в таблицу Documents

    query = `INSERT into Documents (Model, img)

    values (?, ?);`;

    values = [

        requestBody.mark,

        requestBody.img

    ];

    try {

        await connection.query(query, values);

    } catch (error) {

        console.log(error);

        res.status(400).json(error);

        return;

    }

    //отправка ответа

    res.status(200).json({

        succes: true

    });

}

Листинг 9 – контроллер execution

const connection = require('../mode/connection');

exports.getExecutionByMark = async function (req, res) {

    let value = [req.params.mark];

    let query = `SELECT D, L, H, Type\_connect, n\_connect, d\_connect, D1\_connect, D2\_connect, Weight FROM Execution

        WHERE Model = ?`;

    try {

        let results = await connection.query(query, value);

        res.json(results[0]);

    } catch (error) {

        res.status(404).json(error);

    }

}

exports.deleteExecution = async function (req, res) {

    const queryParam = req.query;

    try {

        await connection.query(`DELETE FROM Execution WHERE Execution.Model = ? AND Execution.D = ?`, [queryParam.model, queryParam.DN]);

        res.status(200).json({

            succes: true

        });

    } catch (error) {

        res.status(404).json(error);

    }

}

exports.editExecution = async function (req, res) {

    let requestBody = req.body;

    try {

        await connection.query(`UPDATE execution SET

        L = ?, H = ?,

        Type\_connect = ?,

        n\_connect = ?,

        d\_connect = ?,

        D1\_connect = ?,

        D2\_connect = ?,

        Weight = ?

        WHERE Model=? AND D=?`, [

            requestBody.L,

            requestBody.H,

            requestBody.Type\_connect,

            requestBody.n\_connect,

            requestBody.d\_connect,

            requestBody.D1\_connect,

            requestBody.D2\_connect,

            requestBody.Weight,

            requestBody.Model,

            requestBody.DN,

        ]);

        res.status(200).json({

            succes: true

        });

    } catch (error) {

        res.status(404).json(error);

    }

}

exports.postExecution = async function (req, res) {

    let requestBody = req.body;

    let query = `INSERT into Execution (Execution, Model, D, L, H, Type\_connect, n\_connect, d\_connect, D1\_connect, D2\_connect, Weight)

    values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`;

    let value = [

        `${requestBody.mark}-${requestBody.d}`,

        requestBody.mark,

        requestBody.d,

        requestBody.l,

        requestBody.h,

        requestBody.Type\_connect,

        requestBody.n\_connect,

        requestBody.d\_connect,

        requestBody.D1\_connect,

        requestBody.D2\_connect,

        requestBody.Weight,

    ];

    try {

        await connection.query(query, value);

        res.status(200).json({

            succes: true

        });

    } catch (error) {

        res.status(400).json(error);

    }

}

Листинг 10 – контроллер conditions

const connection = require("../mode/connection");

/\*..conditions/tightness\*/

exports.getTightnessClass = async function (req, res) {

    let results = await connection.query('SELECT Class FROM tightness\_class');

    let tightness = [];

    results[0].forEach(tClass => {

        tightness.push(tClass.Class)

    })

    res.json(tightness);

}

/\*..conditions/climate\*/

exports.getClimateClass = async function (req, res) {

    let results = await connection.query('SELECT Climate FROM storage\_conditions');

    let climate = [];

    results[0].forEach(tClass => {

        climate.push(tClass.Climate)

    })

    res.json(climate);

}